

VUPIC

Virtual Machine Usage Based Placement in IaaS Cloud

Gaurav Somani, *member, IEEE*, Prateek Khandelwal, and Kapil Phatnani

Abstract—Efficient resource allocation is one of the critical performance challenges in an Infrastructure as a Service (IaaS) cloud. Virtual machine (VM) placement and migration decision making methods are integral parts of these resource allocation mechanisms. We present a novel virtual machine placement algorithm which takes performance isolation amongst VMs and their continuous resource usage into account while taking placement decisions. Performance isolation is a form of resource contention between virtual machines interested in basic low level hardware resources (CPU, memory, storage, and networks bandwidth). Resource contention amongst multiple co-hosted neighbouring VMs form the basis of the presented novel approach. Experiments are conducted to show the various categories of applications and effect of performance isolation and resource contention amongst them. A per-VM 3-dimensional Resource Utilization Vector (RUV) has been continuously calculated and used for placement decisions while taking conflicting resource interests of VMs into account. Experiments using the novel placement algorithm: VUPIC, show effective improvements in VM performance as well as overall resource utilization of the cloud.

Index Terms—Cloud Computing, Resource Allocation, Virtual Machine Placement and Scheduling.

I. INTRODUCTION

CLOUD COMPUTING has emerged as a phenomenal technology where computing services are provided over the network, with on-demand elastic resources like computing power, memory, storage and network bandwidth. Virtualization provides an easy solution to the objective of cloud, by facilitating creation of Virtual Machines (VMs) on top of the underlying hardware, or, Physical Machines (PMs), resulting into improved resource utilization and abstraction. Infrastructure as a Service (IaaS) clouds provides raw VMs on which the user can install their own software or applications.

The issues that a cloud service provider must take care of are elasticity, scalability, migration, and performance isolation, out of which, performance isolation is a very critical challenge. The resource allocation algorithms take resource requirement of a VM into consideration and changes the allocated resources, thus making it an on demand elastic cloud. Virtual machine placement and migration are an integral part of resource allocation in cloud. Changing resource requirements of VMs can be a critical information for VM placement and migration decisions. In a cloud, placement algorithms are

responsible for efficient placement of VMs on physical hosts. Initial placement of a VM and subsequent resource usage based placement form a resource allocation procedure in cloud.

In this paper, a novel VM Placement algorithm is presented which takes into account the application specific resource usage made by the VM. The resources are classified into three categories, CPU, network, and disk I/O. Any application would consume these three basic resources. The collective usage of these resources by the applications will result in the total resource usage by the VM. On the basis of usage of these resources, we classify or differentiate various VMs and apply new VM placement algorithm.

The organization of the paper is as follows, section II discusses the problem of performance isolation and its relation with the formulation of VM placement, section III introduces an initial experiment, section IV introduces the novel algorithm, section V discusses the experiment to explore the success to the proposed algorithm, and section VII discusses the current implementation of the proposed algorithm, Section VIII discusses the related work and section IX describes conclusion and future work.

II. ROLE OF PERFORMANCE ISOLATION AND RESOURCE CONTENTION IN VIRTUALIZED ENVIRONMENT

An efficient resource allocation method will optimally place VMs on PMs such that the overall resource utilization is maximized [19]. It also incorporate ways of introducing efficient and green data centres [9], [10], and increase *return on investment (ROI)*. As stated in section I, performance isolation is very critical issue, and has been studied extensively [12], [13], yet an ideal situation of performance isolation cannot be achieved. With some initial experiments to understand the co-existential or neighbour dependent behaviour of VMs, the authors inferred that the problem of performance isolation can be improved by reducing the resource contention amongst the VMs on same physical host.

Performance isolation can be drawn to the lowest level abstraction of shared resources like cpu, memory, network and disk, for example, disk is continuously being used by multiple processes waiting in I/O queue. Thus, I/O scheduler will play a vital role in resource contention, as studied in [14]. To review the resource contention properties of various applications (in turn, VMs), experiment 1 has been conducted and described in next section.

Name	Processor	Memory	Function
alpha	Intel (r)	4 GB	Physical Server 1
beta			Physical Server 2
gamma	Core (TM) i5		Storage Server (NFS)
delta			Physical Server 3

TABLE I
PHYSICAL SERVER DETAILS IN CLOUD

III. EXPERIMENT 1

The objective of experiment 1 is to observe the co-existential behaviour of different VMs, and how their performance is affected when placed with certain kind of VMs. The VMs used in this experiment exploit the individual resources like CPU, Network, and Disk I/O.

A. Experimental Set-up

The set-up of the experiment 1 is as follows, machines alpha, beta and gamma are the physical hosts for the VMs, and machine gamma acts as Network File Server (NFS), to hold the disk images of the VMs (refer figure 1(a)). The configuration for the physical machines is provided in table I. The configuration for the virtual machines is provided in table II.

B. Test VMs

The experiment was conducted by executing the following benchmarks -

- *CPU intensive* : `sysbench` [26] was run in CPU test mode for 120 seconds to calculate the prime numbers from 1 to 80,000. The result generated was the number of events of prime calculation that occurred in 120 seconds.
- *Disk Intensive* : `sysbench` was run in fileIO mode for 120 seconds to perform random read and write operations on 128 files of 2GB, the result generated here too was the number of events that occurred in 120 seconds.
- *Network Intensive* : the Apache HTTP server was installed on the VM, serving a static HTML web page, and was benchmarked as follows, using `ab` [27], to serve 100,000 requests, with 10 concurrent request in each iteration. The result generated here was the average time taken per request, in milliseconds.

The machines CPU1 and CPU2 are cloned instances of CPU intensive test VMs specified in section III-B, similarly for DISK1 and DISK2, and for WEB1 and WEB2.

The experiment is re-run after rearranging the VMs on physical hosts, providing the following combinations of VMs as CPU intensive with Network intensive, CPU intensive with Disk intensive, Network intensive with Disk intensive (refer figure 1(b))

Type	Machines	Initial Host
CPU Intensive	CPU1, CPU2	alpha
Disk Intensive	DISK1, DISK2	beta
Network Intensive	WEB1, WEB2	delta

TABLE II
VM CLASSIFICATION

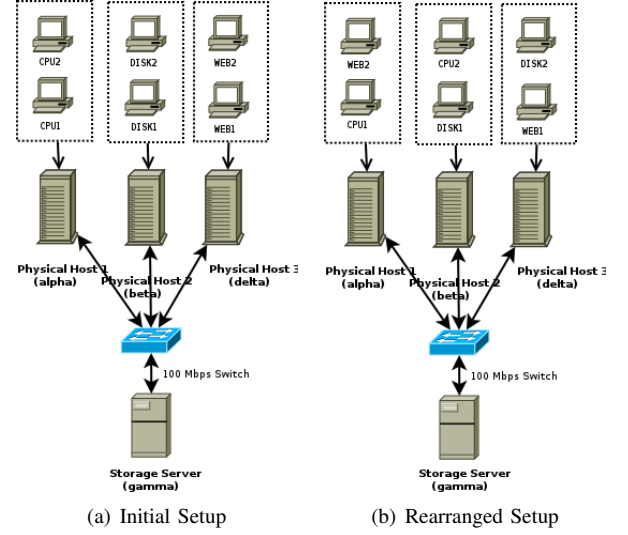


Fig. 1. Setup for Experiment 1

C. Results for the Experiment 1

The results for experiment 1 are provided in tables III, IV, and V. The Tables compares the performances of a particular type of machine, when it is paired with a CPU intensive, Disk Intensive, and Network Intensive machine respectively. The rearrangement as shown in figure 1(b) provides for all the possible combinations of VMs and provides a base for comparisons. The results for CPU and Disk Intensive machines were number of events that occurred in the given time, while that for Network Intensive machine was the average time taken for completion of the HTTP requests.

-	CPU	Disk	Network
CPU (No. of Events)	5636	7803	5845

TABLE III
RESULTS OF EXPERIMENT 1 FOR CPU INTENSIVE MACHINE

-	CPU	Disk	Network
Disk (No. of Events)	3900	3000	4400

TABLE IV
RESULTS OF EXPERIMENT 1 FOR DISK INTENSIVE MACHINE

-	CPU	Disk	Network
Network (Average Response Time)	12.653ms	15.329ms	21.281ms

TABLE V
RESULTS OF EXPERIMENT 1 FOR NETWORK INTENSIVE MACHINE

In table III, the increase in performance is significant when kept with a disk intensive VM, which can be attributed to the fact that it will consume a minimal level of CPU time. Similarly, in table IV, the performance of a disk intensive VM is compromised when kept with another disk intensive VM, and similar is the case for a network intensive VM, as shown in V. The results show that for any VM, when kept with another VM that extensively uses the similar kind of resource, the performance drops significantly.

D. Discussion

The resource usage patterns of each of the VM type can be represented as a three dimensional vector, which will be called as *Resource Usage Vector (RUV)* in rest of the paper. The components of *RUV* are *cpu*, *network*, and *disk I/O*, where each component can take value of *L, M, H* and *L* represents *low* usage, *M* represents *medium* usage and *H* represents *High* usage of the particular component or resource.

$$\vec{RUV} = \langle \text{CPU usage}, \text{Network usage}, \text{Disk I/O usage} \rangle \quad (1)$$

While running the applications, the resource usage logs were generated for each VM for the entire run of 120 seconds, where the CPU usage, the network usage and the Disk I/O usage for the last 2 seconds per iteration were logged. Using these logs, mean was calculated for each machine for each component (mean is adopted here just for the sake of simplicity, and can be replaced by any other function like maximum magnitude, mean with standard deviation, mean after curve smoothing etc.). Thus, by setting appropriate range for the mean utilization data, resource vector for the machines came out to be:

$$\begin{aligned} \text{CPU Intensive} &= \langle H, L, L \rangle \\ \text{Network Intensive} &= \langle M, H, L \rangle \\ \text{Disk Intensive} &= \langle L, L, H \rangle \end{aligned}$$

For the experiment, the utilization range for the resources as mentioned in table VI were followed. (Note - the limit ∞ is used to represent the maximum upper limit.)

-	L	M	H
CPU Usage(%)	[0,20)	[20,70)	[70,100]
Network Usage(B/sec)	[0,10K)	[10K,800K)	[800k, ∞)
Disk I/O Usage (B/sec)	[0,10K)	[10K,200K)	[200K, ∞)

TABLE VI
RANGE FOR THE COMPONENTS OF RESOURCE USAGE VECTORS

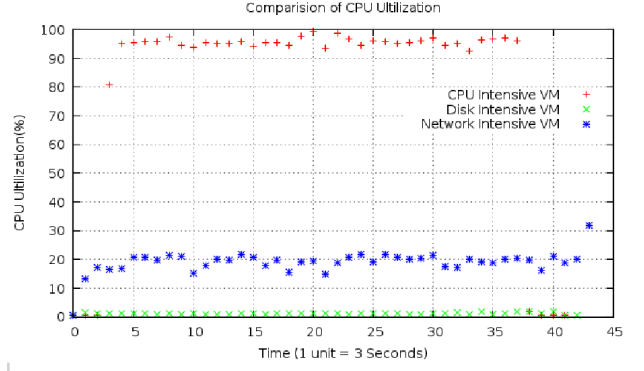


Fig. 2. Comparison of CPU Utilization

The CPU utilization of different VMs can be seen in Fig. 2. Average utilization of CPU Intensive VM remains around 80%. The Average CPU utilization of Network Intensive VM, which is a web server, gives around 20% CPU utilization, as it serves multiple request at a time, while that of a Disk intensive VM is zero. The lower limit for High CPU Utilization (H) is set at 70% to incorporate any minor fluctuations that may occur, and bring down the average. For the lower limit of Medium CPU utilization, 20% is used by taking the assumption that a Network Intensive VM will exhibit medium CPU utilization as in the current scenario, since not much server side processing is been done, the CPU utilization is bound to be above this level in any real scenario.

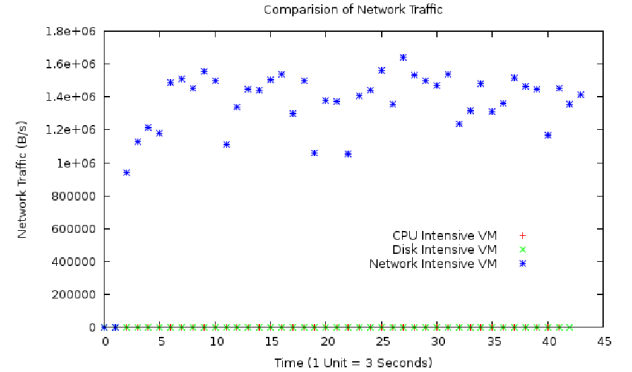


Fig. 3. Comparison of Network Traffic

The Network traffic by different VMs is given in Fig. 3. Here, for both CPU intensive VM, and for Disk Intensive VM, the Network traffic is almost zero, hence the lower limit for medium Network traffic is taken as 10KB/s to be on the safer side. The lower limit for High Network Traffic is taken at around 800KB/s.

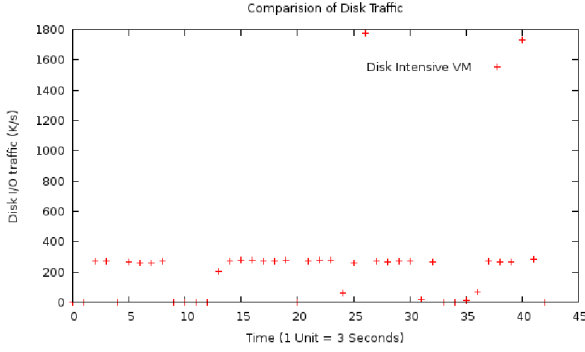


Fig. 4. Comparison of Disk Traffic

The Disk traffic for CPU intensive and for the Network Intensive machines are almost zero, except for a few fluctuations (not marked in Fig. 4), therefore, considering the fact that Disk Intensive VM will exhibit high Disk traffic, the lower limit for High Disk traffic (H) is kept at 200 K/s (bytes). For lower limit of Medium Disk traffic, 10 K/s is used to accommodate any minor jitters (refer Fig. 4).

The experiment gives an insight on the performance isolation provided by Xen, here it is clearly evident that resource contention results in performance drop, whether in the case of CPU intensive, Disk intensive, or Network intensive applications. It is important note that these resulting RUVs are generated after running benchmarks in stressed modes, however, real enterprise applications, like web servers, often run under normal mode, and hence the RUV may change accordingly. However, it is also clearly evident that placing a VM with another VM, which is using different type of resource, and hence having its vector *orthogonal* to that of the former VM, results in mutual benefit for both.

In short, in order to have least amount of resource contention amongst VMs, the probability that two VMs which extensively use similar kind of resources be placed on the same host should be minimized. The authors exploit this *Co-existential behaviour* in the following part of the paper.

Note: The condition that the VMs on a particular physical server so not exceed the available resources is assumed implicit to the algorithm, and can be easily incorporated in the production environment

IV. THE VUPIC ALGORITHM

A. Motivation

The results obtained in experiment 1 show that VMs perform better in absence of resource contention. The RUV mentioned in equation 1 provides an efficient method of quantifying the resource usage across different resources and can be employed in placement decisions. The proposed algorithm rearranges the VMs according to their RUVs. The main objective of VUPIC is to club together those VMs which consume different resources, so as to minimize resource contention and also exploiting the resources available. VUPIC also ensures that no to VMs which extensively use same kind of resources are placed together, unless the situation can't be avoided. However, it will place two VMs using same resources if one of the VM uses the resource less extensively or normally.

The novel algorithm applies an optimization while making rearrangements on the basis of VMs corresponding RUVs. VUPIC would start arranging the VMs on PMs with a priority assignment that in the new arrangement, if possible, the VM should remain on the original host, in order to save VM migration costs. This will occur when the VUPIC is filling the PMs (bins) with the VMs and if the original or earlier PM is capable enough of holding the VM by satisfying the resource contention/ performance isolation requirements, the VM should remain on the original host so that one VM migration migration can be saved. This optimization might save many VM migration costs. This is applied as mentioned in line no. 11 in procedure 1.

There might occur a case where a VM may not find any suitable host according to the proposed method, such a VM will be referred to as *Compromised VM*. VUPIC provides a data structure “compromisedVMList” which stores all such compromised VMs. After allocation of all the VMs according to the algorithm, VUPIC places the *Compromised VMs* in a manner such that the performance loss is minimized (refer Line no. 26 - 31 in procedure 1). For example, assume two physical host, M1 and M2, and three VMs V1, V2 and V2 with RUVs (H, M, L), (M, H, L), and (M, H, L). VUPIC will place V1 on M1, and V2 on M2. Now, according to the algorithm, V3 will be placed along with V2, as CPU has more priority than network, and since V2 is hosted on M2, V3 will be placed on M2.

The present algorithm can easily work with algorithms designed to manage the elasticity of the resources owned by the VMs. After rearranging the VMs according to the proposed algorithm, the RUV may change, i.e., it may either increase or decrease, as the previous performance of machine could have been suppressed or compromised due to resource contention with similar VMs.

B. Invocation of VUPIC

The appropriate time of invocation of VUPIC depends on the user. One way would be to regularly call VUPIC after fixed duration of time and generate the placement schedule. Other way is to invoke the VUPIC when a client up-scales or down-scales its resource requirement and corresponding component of RUV changes. VUPIC can also be coupled with a resource manager which manages the scalability requirements of VMs on physical hosts, and in such case, VUPIC will be invoked if even after scaling the VM on the host, its resource vector does not change for a particular amount of time.

Procedure 1 Virtual Machine Usage Based Placement in Iaas Cloud (VUPIC)

Input: 1) Set of RUVs $\langle C, N, D \rangle$ of each VM (MachinePool)
 2) present placement configuration of VMs
 Output: 3) New placement configurations of VMs

```

00: BEGIN
01: Sort(MachinePool, DESC, vm.RUV)
02: compromisedVM = []
03: for all physicalHosts{
04:   set physicalHosts.state = < H, H, H >
05: }
06: }
07: while MachinePool is not Empty {
08:   placed = no
09:   VM = MachinePool.remove()
10:   Sort(PhysicalHosts, DESC, PhysicalHost.state)
11:   place VM's original host in front of the queue
12:   for all PM in physicalHosts{
13:     if (PM.state > VM.ruv){
14:       place(VM, PM)
15:       placed = yes
16:       break
17:     }
18:   }
19:   if (placed = no){
20:     for all PM in physicalHosts:
21:       if (PM.state == VM.ruv){
22:         place(VM, PM)
23:         placed = yes
24:         break
25:       }
26:   }
27:   if (placed = no){
28:     compromisedVM.insert(vm)
29:   }
30: }
31: while compromisedVMList is not Empty {
32:   vm = compromisedVMList.remove()
33:   Sort(PhysicalHosts, DESC, PhysicalHost.state)
34:   place(vm, pm)
35: }
36: }
37: function place(vm, pm):
38:   for j = 1 to 3 : {
39:     pm.state[j] = pm.state[j] - v.ruv.[j]
40:   }
41: }
42: function Sort(List, order, value)
43:   # Sorts the given list
44:   # by the specified order and value
45: END

```

V. EXPERIMENT 2(A) : VUPIC

A. Experimental set-up

The applications that were used in experiment 1 only intensively exploits the basic resources. The modern data centres and cloud host applications where the HPC and web server resemble the VMs used in experiment 1, hence to account for this property of cloud, two VMs which run Database server are added to the cloud. The VM classification for experiment 2(a) is as specified in table VII, and in figure 5(a). The machines CPU1 and CPU2 are cloned instances of CPU intensive test VMs specified in section III-B, similarly for DISK1 and DISK2, for WEB1 and WEB2, and for DBMS1 and DBMS2.

B. Tests

The experiment was conducted by executing the following benchmarks:

- *CPU Intensive* : sysbench was run in CPU test mode for 120 seconds to calculate the prime numbers from 1

Type	Machines
<i>HPC</i>	CPU1, CPU2
<i>Disk I/O</i>	DISK1, DISK2
<i>HTTP Server</i>	WEB1, WEB2
<i>DBMS server</i>	DBMS1, DBMS2

TABLE VII
VM CLASSIFICATION FOR EXPERIMENT 2(A)

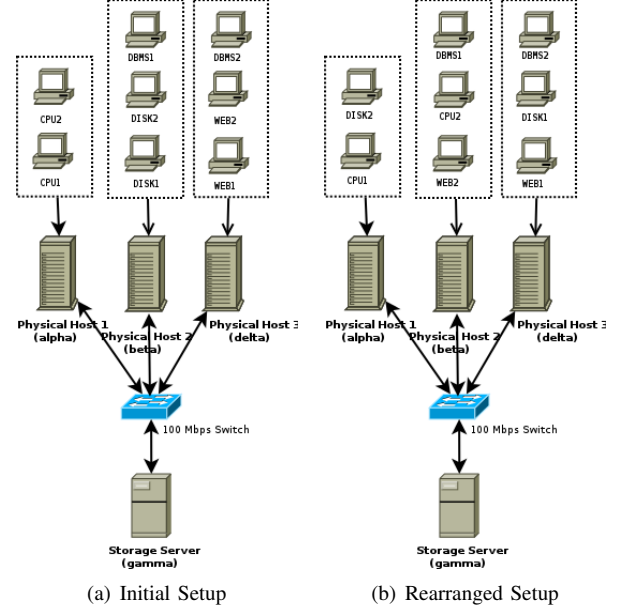


Fig. 5. Setup for Experiment 2(a)

to 80,000. The result generated was the number of events of prime calculation that occurred in 120 seconds.

- *Disk Intensive* : sysbench was run in fileIO mode for 120 seconds to perform random read and write operations on 128 files of 2GB, the result generated here too was the number of events that occurred in 120 seconds.
- *Network Intensive* : the Apache HTTP server was installed on the VM, serving a static HTML web page, and was benchmarked as follows, using ab, to serve 50,000 requests with 20 concurrent request in each iteration. The result generated here was the total time taken by the test, in seconds.
- *DBMS server* : sysbench was run in OLTP mode for 120 sec, performing a series of write operations, the result generated was the number of successful transactions.

C. The Resource Usage Vector (RUV)

The resource usage vector was calculated by same method as mentioned previously in subsection III-D, and the ranges for specifying L, M and H for each component of CPU, Network, and Disk are as specified in table VI. The generated RUVs are mentioned in table VIII.

Machine	CPU	Network	Disk I/O
CPU1 (CPU)	H	L	L
CPU2 (CPU)	H	L	L
DISK1 (Disk I/O)	L	L	H
DISK2 (Disk I/O)	L	L	H
WEB1 (HTTPd)	M	H	L
WEB2 (HTTPd)	M	H	L
DBMS1 (MySQL)	L	L	H
DBMS2 (MySQL)	L	L	H

TABLE VIII
RESOURCE USAGE VECTORS FOR THE VMS

The setup after considering the resource usage vectors (refer table VIII) and placing the machines according to the algorithm, the new rearrangement was generated. The change in the placement is shown in table IX, and in figure 5(b).

Machine	Initial Host	New Host
CPU1	alpha	alpha
CPU2	alpha	beta
DISK1	beta	delta
DISK2	beta	alpha
WEB1	delta	delta
WEB2	delta	beta
DBMS1	beta	beta
DBMS2	delta	delta

TABLE IX
HOSTS FOR THE VMS

D. Results of Experiment 2(a)

Machine	Performance (before)	Performance (after)
CPU1	4286#	5527#
CPU2	4812#	5199#
DISK1	1466#	2830#
DISK2	2200#	5600#
WEB1	136s	71.367s
WEB2	136s	94s
DBMS1	208#	725#
DBMS2	686#	365#

TABLE X
COMPARISON OF RESULTS OBTAINED IN EXPERIMENT 2(A) BEFORE AND AFTER RE-ARRANGEMENT OF VMS

All the performances are given in terms of number of events completed by the test, except for web servers, where it stands for time taken to complete the test (in seconds).

The comparison of the results obtained is provided in table X. Here, the mutual benefit that is provided to VMs after rearranging them according to the proposed algorithm is very promising. For the case of the VMs CPU1 and CPU2, while their performance before applying the algorithm was 4286 and 4812 events respectively, upon relocating CPU1 with DISK2, the performance of CPU1 rises to 5527 events, for CPU2

(relocated with DBMS1 and WEB2), the performance rises to 5199 events.

Similarly, if we observe the machines originally placed on physical host - *beta*, the performance of DISK1 rises from 1466 to 2830 events when placed along with DBMS2 and WEB1, so does for DISK2, where its performance rises from 2200 events to 5600 events on placing with CPU1, this more than twofold increase in case of DISK2 can be attributed to the absence of disk intensive VMs on physical host *alpha*, which is not the case for DISK1, where one disk intensive VM is already present (DBMS2 on *delta*). For the case of VM DBMS1, its performance also increases as from 208 to 725 transaction (more than three fold increase), upon relocation of DISK1 and DISK2 from *beta*.

For the machines on physical host *delta*, the performance of VM WEB1 becomes double, as its response time is reduced to almost half (from 136 seconds to 71 seconds) when placed with DISK1 and DBMS2, similarly for WEB2, the performance is increased from 136 seconds to 94 seconds, upon its relocation on physical host *beta*, with DBMS1 and CPU2. The case of compromised placement is also visible here, where the performance of DBMS2 decreases to its half, i.e., drops from 686 events to 365 events, and the difference in the increase that DISK1 and DISK2 experience (increase by 1364 events in case of DISK1, while increase by 3400 events for DISK2) can be seen as the side-effect caused by placing the compromised VM DBMS2 on machine *delta*, which also hosts DISK1 (both DISK1 and DBMS2 share extensively use Disk I/O, hence resource contention comes into picture again).

VI. EXPERIMENT 2(B) - VUPIC WITH MULTIPLE VCPU CONSIDERATIONS

In real world, most of the applications are multi-threaded, and also may have more than one VCPU. It is recommended to keep the number of VCPUs equal to that of PCPUs [20], experiment 2(b) is presented to show the capability of the algorithm to take the fact into account that a VM may have more than one VCPUs. Most of the applications in cloud are multi-threaded, like web server and DBMS, A modified version of the proposed algorithm is given in figure VI-B which will also factor the number of VCPUs allotted to a VM. For any physical host, there cant be more VCPUs present on it than the number of PCPUs available, and the same is formulated in subsection VI-A.

A. Upper Limit on VCPUs

In this version of algorithm, zero CPU overcommitment policy is followed as it is recommended by Xen enterprise server vendors [20]. Therefore the mathematical formulation for this case will be,

$$\forall j \text{ Such That } PM_j \text{ is a Physical Server}$$

$$\sum_{VM_i \in VM \text{ on } PM_j} VCPU_{vm_i} \leq \text{number of PCPUs on } PM_j \quad (2)$$

However, the overcommitment policy can be changed by simply increasing the PCPU limit of a physical host.

B. Experimental setup

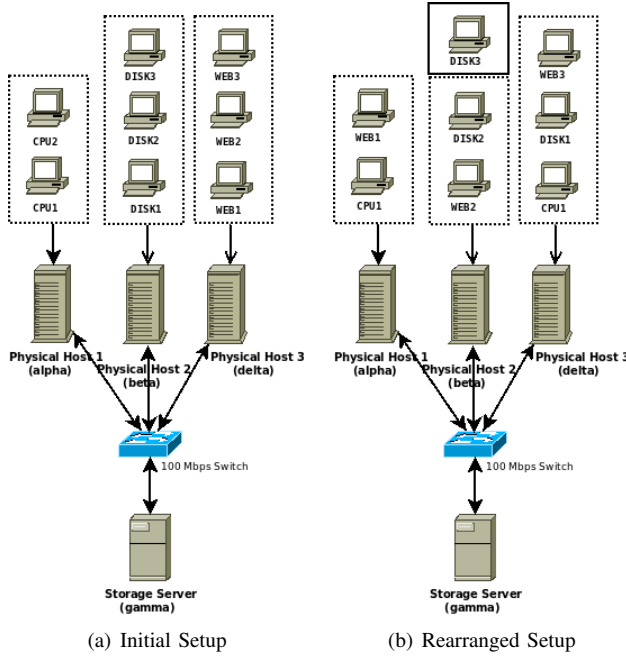


Fig. 6. Setup for Experiment 2(b)

The initial VM arrangement is same as that in experiment 2(a) (refer figure 6(a)), with the role of DBMS1 changed to that of a disk intensive VM. It will be called DISK3. Similarly, the role of DBMS2 changed to a web server (named WEB3), and each VM may have one or two VCPUs (refer table XI). The benchmarks used are same as mentioned in section III-B), with the exception of CPU2, DISK2, DBMS1, WEB2, DBMS2, where each will use two threads (refer table XI).

Procedure2: VUPIC (with CPU overcommitment constraints)

```

Input: 1)Set of RUVs <C,N,D> of each VM along with
       their VCPU requirement (MachinePool)
       2)present placement configuration of VMs
Output: 1)New placement configurations of VMs
00: BEGIN
01: Sort(MachinePool, DESC, vm.RUV)
02: compromisedVM = []
03: for all physicalHosts{
04:   set physicalHosts.state = < H, H, H >,
05:   physicalHosts.PCPU = # number of processors available
06: }
07: while MachinePool is not Empty {
08:   placed = no
09:   VM = MachinePool.remove()
10:   Sort(PhysicalHosts, DESC, PhysicalHost.state)
11:   place VM's original host in front of the queue
12:   for all PM in physicalHosts{
13:     if (PM.state > VM.ruv) and (PM.pcpu >= VM.vcpu){
14:       place(VM,PM)
15:       placed = yes
16:       break
17:     }
18:   }
19:   if (placed = no){
20:     for all PM in physicalHosts:
21:       if (PM.state == VM.ruv) and (PM.pcpu >= vm.vcpu){
22:         place(VM,PM)
23:         placed = yes
24:         break
25:       }
26:   }
27:   while compromisedVMList is not Empty {
28:     vm = compromisedVMList.remove()
29:     Sort(PhysicalHosts, DESC, PhysicalHost.state)
30:     placed = no
31:     for all pm in PhysicalHosts{
32:       if (pm.pcpu >= vm.vcpu){
33:         place(vm,pm)
34:         placed = yes
35:       }
36:     }
37:     if (placed == no)
38:       print "VM ",vm, "INVALID/UNFIT"
39:   }
40:   function place(vm,pm):
41:     for j = 1 to 3 : {
42:       pm.state[j] = pm.state[j] - v.ruv.[j]
43:     }
44:     pm.pcpu = pm.pcpu - vm.vcpu
45:   }
46:   function Sort(List, order, value)
47:     # Sorts the given list
48:     # by the specified order and value
49:   }
50: END

```

Fig. 7. VUPIC (with CPU overcommitment constraints)

Machine	Initial Host	VCPUs and Threads
CPU1	alpha	1
CPU2	alpha	2
DISK1	beta	1
DISK2	beta	2
DISK3	beta	2
WEB1	delta	1
WEB2	delta	2
WEB3	delta	2

TABLE XI
HOSTS FOR THE VMs AND VCPUs ALLOTTED

C. Results of Experiment 2(b) and Discussion

The generated migration schedule is (table XII):

Machine	Initial Host	New Host
CPU1	alpha	delta
CPU2	alpha	alpha
DISK1	beta	delta
DISK2	beta	beta
DISK3	beta	INVALID/UNFIT
WEB1	delta	alpha
WEB2	delta	beta
WEB3	delta	delta

TABLE XII
HOSTS FOR THE VMS AND VCPUS ALLOTTED

After running the experiment as mentioned above, the VM DISK3 was not assigned to any physical host (refer figure 6(b)), as it requires 2 VCPUs and assigning it to any physical server would breach the PCPU limit. In such a case VUPIC declares such a VM to be INVALID/UNFIT for placement. This behaviour is enforced so that the other VMs may not be affected by the high need of VCPUs by DISK3, however, it can be placed with a high degree of compromise, or can be migrated to another cloud, and this decision can be made by the user and can be configured in VUPIC.

VII. IMPLEMENTATION OF VUPIC

In current version, VUPIC is a collection of interdependent python scripts, which handles the task of logging the resource usage done by the VMs, producing RUVs using the mean resource usage from the generated log, and generating placement schedule. The VUPIC clients generate the resource usage logs and using them generate the RUVs as mentioned in equation 1, and the ranges for the components used are as specified in table VI. The resource generation script has a CPU overhead of around 3%, and can be attributed to its collection of information from three different sources of xentop [28], ifstat [29], and iotop [30]. These ranges can be changed with ease, and as mentioned before, the statistical function used to generate the RUV can be replaced by any other function.

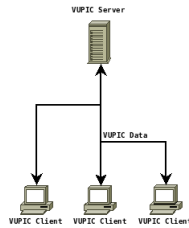


Fig. 8. VUPIC Architecture

The current version of the VUPIC client generates the RUVs and stores them on the storage server for each client as a file. The VUPIC server uses these files to generate the placement schedule. This schedule can be used by any migration manager to execute the migration.

VIII. RELATED WORK

The problem of resource allocation is similar to that in Grids and Clusters. Some of the popular resource allocation managers

used in various environment are GRAM [21], and Condor [22]. Some popular cloud management system are Open Nebula [23], Eucalyptus [24] and Nimbus [25]. OpenNebula provides for *Match Making Scheduler* where it does *rank scheduling* as specified by the user. Resource allocation issue also arises when a VMs upscale their available resource to meet the increased demand. Virtual machine placement is a critical and central issue related to resource allocation in cloud and it has been studied extensively. C. Hyser et. al. [3] provides with a framework for autonomic placement manager. There are a number of virtual machine placement algorithms which aim to provide energy efficient scheduling, A. Verma et. al. [9] have studied the problem of power consumption in cloud and have proposed pMapper in [2]. B. Li et. al. proposed enaCloud in [8] which aims to achieve similar objective, another such research is done by J. Xu et. al. in [4] which along with power, also takes into account thermal dissipation costs. Similar work is done by A. Beloglazov et. al. in [10], and in [6]. When looking at the VM placement problem from the point of view of maximizing the available physical resource usage, performance isolation becomes another critical issue that needs to be taken care of. J. N. Matthews et. al. provide a design for performance isolation benchmark in [1]. The performance isolation provided by Xen and the effect of I/O intensive applications with low latency has been studied in [12]. As the performance isolation can be mapped to low level of physical machine, the CPU scheduler also plays a role in deciding performance of virtual machines as studied in [14]. There are a number of resource allocation strategies which have been created from the point of view of maximizing physical resource utilization as studied in [11], the problem reduces to that of the classical NP-hard problem of bin packing. Another approach is to reduce the network traffic due to intra-VM communications as studied in [16]. Network aware placement has been studied by J. T. Piao et. al. in [5]. These algorithms work with optimal migration algorithm, which may take into account the post-migration network load [17] and minimizing the VM down time [7]. The proposed novel VM placement algorithm significantly improves the VM performance, which otherwise was compromised. The paper also introduces the concept of resource usage variation and provides efficient method to incorporate the same using Resource Usage Vectors (RUV) in placement decision. The algorithm generates a placement schedule, which can be easily deployed using any of the above mentioned migration techniques.

IX. CONCLUSION AND FUTURE WORK

The present work incorporates performance isolation and resource contention properties into account while taking VM placement as well as resource allocation decisions. Any infrastructure cloud would be a multi-tenant service provider and would host virtual machines of varied types on different physical servers. Virtual machines generate different resource usage and thus create a behavioral usage pattern. Experiments were conducted to show how individual VMs affect and get affected by the neighboring VMs on the same physical server. Currently, there is no virtual machine placement algorithm that takes the resource usage patterns into account.

The proposed novel algorithm takes resource usage by the virtual machines into account while making placement decisions, and also provides an efficient way to incorporate these resource usage patterns into the algorithms using a 3-dimensional vector called Resource Usage Vector (RUV). A modified version of the same algorithm has been presented which takes care of VMs with multiple VCPUs and also restricts the CPU over commitment. Both of these algorithms generate a migration schedule which minimizes the number of unnecessary migrations by allocating a VM to its original host if all the constraints are satisfied. The generated migration schedule can be executed using an optimal migration algorithm to reduce the migration costs. Authors also plan to incorporate power efficiency and migration algorithms to implement it as a complete IaaS solution

REFERENCES

- [1] J. N. Matthews, W. Hu, M. H., T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, J. Owens, "Quantifying the Performance Isolation Properties of Virtualization Systems", *ExpCS07*, 2007
- [2] A. Verma, P. Ahuja, A. Neogi, "pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems", *Middleware 2008*, pp 243-264, 2008
- [3] C. Hyser, B. McKee, R. Gardner, B. J. Watson, "Autonomic Virtual Machine Placement in Data Center", *HP Laboratories, HPL-2007-189*, 2008
- [4] J. Xu, J. A. B. Fortes, "Multi-objective Virtual Machine Placement in Virtualized Data Center Environments", *IEEE/ACM International Conference on Green Computing and Communications*, pp 179-188, 2010
- [5] J. T. Piao, J. Yan, "A Network-aware Virtual Machine Placement and Migration Approach in Cloud Computing", *Ninth International Conference on Grid and Cloud Computing*, pp 87-92, 2010
- [6] K. Le, J. Zhang, R. Bianchini, Y. Jaluria, J. Meng, T. D. Nguyen, "Reducing Electricity Cost Through Virtual Machine Placement in High Performance Computing Clouds", *ACM SC 11*, 2011
- [7] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, "Live Migration of Virtual Machines", *NSDI 05: 2nd Symposium on Networked Systems Design & Implementation*, pp 273-286, 2005
- [8] B. Li, J. Li, J. Huai, T. Wo, Q. Li, L. Zhong, "EnaCloud: An Energy-saving Application Live Placement Approach for Cloud Computing Environments", *IEEE International Conference on Cloud Computing*, pp 17-24, 2009
- [9] A. Verma, P. Ahuja, A. Neogi, "Power-aware Dynamic Placement of HPC Applications", *JCS'08*, pp 175-184, 2008
- [10] A. Beloglazov, R. Buyya, "Energy Efficient Resource Management in Virtualized Cloud Data Centers", *CCGRID '10 Proceedings of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, 2010
- [11] U. Bellur, C. Rao, Madhu Kumar S. D., "Optimal Placement Algorithms for Virtual Machines", *Arxiv preprint arXiv:1011.5064*, 2010
- [12] G. Somani, S. Chaudhary, "Application performance Isolation in Virtualization", *IEEE International Conference on Cloud Computing, CLOUD '09*, 2009
- [13] D. Gupta, L. Cherkasova, R. Gardner, A. Vahdat, "Enforcing performance isolation across virtual machines in Xen", *Middleware '06 Proceedings of the ACM/IFIP/USENIX International Conference on Middleware*, 2006
- [14] L. Cherkasova, D. Gupta, A. Vahdat, "Comparison of three CPU Schedulers in Xen", *Performance Evaluation Review - xen.org*, 2007
- [15] C. P. Sapuntzakis, R. Chandra, B. Pfaff, J. Chow, M. S. Lam, M. Rosenblum, "Optimizing the migration of virtual computers", *ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, 2002
- [16] S. Sudevalayam, P. Kulkarni, "Affinity-aware Modelling of CPU Usage for Provisioning Virtualized Applications", *IEEE International Conference on Cloud Computing (CLOUD)*, 2011
- [17] V. Shrivastava, P. Zeros, K. Lee, H. Jamjoom, Y. Liu, S. Banerjee, "Application-aware Virtual Machine Migration in Data Centres", *IEEE INFOCOM*, pp 66-70, 2011
- [18] F. Chang, J. Ren, R. Viswanathan, "Optimal Resource Allocation in Clouds", *IEEE 3rd International Conference on Cloud Computing*, pp 1-8, 2010
- [19] P. Sargeant, "Data Centre Transformation: How Mature is your IT", *Presentation by Managing VP, Gartner*, Feb 2010
- [20] Virtualization Best Practices for XenApp, Citrix Systems Inc. <http://blogs.citrix.com/2011/06/15/virtualization-best-practices-for-xenapp/>
- [21] Grid Resource Allocation Manager (GRAM), <http://dev.globus.org/wiki/GRAM>
- [22] Condor project, <http://research.cs.wisc.edu/condor/>
- [23] openNebula open-source cloud management system, <http://opennebula.org/>
- [24] Eucalyptus open-source cloud management system, <http://open.eucalyptus.com/>
- [25] Nimbus Cloud, <http://www.nimbusproject.org/>
- [26] Sysbench benchmarking suite, <http://sysbench.sourceforge.net/>
- [27] Apache HTTP server benchmarking tool, <http://httpd.apache.org/docs/2.0/programs/ab.html/>
- [28] Xentop, Domain monitoring tool for Xen Hypervisor, <http://linux.die.net/man/1/xentop>
- [29] Ifstat, Network Bandwidth Reporting tool, <http://gael.roualland.free.fr/ifstat/>
- [30] Iotop, Disk I/O Usage Analysis tool, <http://guichaz.free.fr/iotop/>

Gaurav Somani is a faculty member at The LNM Institute of Information Technology, Jaipur, India. His research interest include cloud computing, virtualization, ad-hoc networks and distributed computing. He has published in many reputed conferences like IEEE CLOUD 2009, IC3 2010, IEEE PDGC 2010. A monograph has been published on his recent works by VDM publishers on scheduling and isolation in virtualisation.

Prateek Khandelwal is a student at The LNM Institute of Information Technology, Jaipur, India.

Kapil Phatnani is a student at The LNM Institute of Information Technology, Jaipur, India.